AD-A233 496

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKING
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: The Regents of the University of California	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION: Cognitive Science Program, Office of Naval Research (Code 1142PT)	
6c. ADDRESS (City, State, and ZIP Code): University of California, Los Angeles, Office of Contracts and Grants Administration, Los Angeles, California 90024	7b. ADDRESS (City, State, and ZIP Code): 800 North Quincy Street, Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Defense Advanced Research Projects Agency	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014 86-K-0395	
8c. ADDRESS (City, State, and ZIP Code): 1400 Wilson Boulevard, Arlington, VA 22209-2308	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61153N	RR04206	RR04206-0C	442c022

11. TITLE (Include Security Classification)

The Evaluation of Expert System Shells

12. PERSONAL AUTHOR(S)

Novak, John R.; Baker, Eva L., & Slawson, Dean A.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Research	FROM 9/1/86 TO 1/31/91	January 1991	38

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial intelligence, expert system shells, knowledge engineering
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper documents efforts to implement the Artificial Intelligence Measurement System (AIMS) model for expert system shells. The report explains the strategy, provides a review of the literature, and summarizes efforts of a case study of knowledge engineering, first using M-1 and then NEXPERT. The problem domain selected was intended to be a well-structured problem, the selection of an appropriate reliability index for criterion referenced tests.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Susan Chipman	22b. TELEPHONE (Include Area Code): (703) 696-4318 / 22c. OFFICE SYMBOL: ONR 1142CS

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

Project Report #18

THE EVALUATION OF EXPERT SYSTEM SHELLS

John R. Novak

Eva L. Baker

Dean A. Slawson

Center for Technology Assessment
UCLA Center for the Study of Evaluation

January 1991

Artificial Intelligence Measurement System
Contract Number N00014-86-K-0395

Principal Investigator: Eva L. Baker

Center for Technology Assessment
UCLA Center for the Study of Evaluation

This research report was supported by contract number N00014-86-K-0395 from the Defense Advanced Research Projects Agency (DARPA), administered by the Office of Naval Research (ONR), to the UCLA Center for the Study of Evaluation. However, the opinions expressed do not necessarily reflect the positions of DARPA or ONR, and no official endorsement by either organization should be inferred. Reproduction in whole or part is permitted for any purpose of the United States Government.

Introduction

Expert system shells are development tools focused on simplifying the programming requirements for system developers. They consist of standard methods for representing knowledge for given tasks and a standard inference approach. The evaluation of expert system shells presents a challenging set of methodological problems. This paper describes an analysis of the literature and an example of a case study exemplifying a particular evaluation approach.

Background

The evaluation of such shells can be conducted from a descriptive perspective, analyzing and documenting knowledge representation and inference strategies as well as data, experience and other requirements. From user perspectives, a prediction must be made based on this descriptive information. The users choose shells by matching their problems to the configurations of strategies, requirements, and costs of various shells. Such an evaluation strategy may be extremely useful for a particular decision maker selecting among alternative expert system shells, and yet of limited value for a user with a different set of problems and resources. The strategy focuses on the prediction of effectiveness rather than the documentation of effectiveness. Effectiveness can be documented in terms of user satisfaction (his/her perception that the software provided satisfactory support), or by expert inspection of the outcome; that is, the analysis of the resulting implementation.

When the evaluation issue is turned in earnest to the topic of effectiveness, the performance of the resulting system must be assessed. In exploring the utility of evaluation methodology for expert system shell performance, a first option is to study how well such tools meet the expectations of users and/or claims of system developers. An obvious approach to the study of these issues is to conduct an experimental study in which expert system developers are randomly assigned to use shells for particular problems. To be useful, a significant number of programmers would be subjects of the study, archetypal

problems would be posed, and resulting expert systems would be analyzed in terms of their success in solving the tasks intended. From a methodological perspective, such a study would, at a minimum level, need to control for variables such as the following: task complexity, time, programmer experience, and domain knowledge. The experimental context could investigate how the subject programmers represent domain knowledge or study the impact of explicit differences in roles of the knowledge engineer and programmer as they interact with an expert. A repeated measures design, where individual programmers use a series of different shells for a set of experimental problems would require controls for order as well. If the research were also to address generalizability, that is, the utility of a shell across a coherent set of problems for different levels of programmer expertise and support, particular methodological complexities, including counterbalancing tasks within shells would be required.

The feasibility of such a study was considered early in the Artificial Intelligence Measurement System (AIMS) project and rejected for a number of reasons. First, the expert system shell was only one of a number of types of intelligent systems the study was exploring. Second, the logistics and anticipated attrition of programmers for extended research of this sort argued against this strategy, particularly because we were convinced that problems in the experiment needed to be real, complex, and therefore time consuming. "Toy" problems would never provide the verisimilitude of the problem contexts needed for useful studies. Finally, cost would be prohibitive for the time of experts, knowledge engineers, programmers, software and hardware (Korf, 1986).

Instead, the project undertook to survey the existing literature for cues for evaluation and to employ an intensive, qualitative methodology to further our understanding of how expert system shells may be evaluated. In the original design for this component of AIMS, a two by two design was proposed to guide case study analysis. The design was to contrast a well-structured problem and an ill-structured problem using alternative expert system shells and heavy duty (e.g., ART, KEE) vs PC expert system shells. It was intended that we would study the process using observation, interview and analysis of the resulting products. The actual studies focused on a well-structured problem, first using M1™ and then NEXPERT™ in a Macintosh environment.

Review of the Literature

Sources. References were obtained through various sources. Computerized library database searches were performed on three databases using the following descriptors: expert system/ shell or tool in conjunction with perform/ measure/ or assess/ as descriptors, resulting in 81 references. Other references were obtained through the use of Orion, UCLA's online catalog system, and Melvyl, the online catalog of the holdings of the entire University of California library system. Indexes to journals in the areas of expert systems and artificial intelligence were examined, and bibliographies from articles on the topic of expert systems were used to generate further references. Most of the references obtained either dealt with evaluation as a passing issue or were descriptions of a particular product, while others furnished general descriptions of the features of a wide range of expert system shells; only eleven of the sources dealt with the issue of evaluation in a more general sense.

Of these eleven articles, two dealt with general methods and metrics for evaluating software (Francis, 1981; Lynch, 1981), four attempted to develop standards for evaluation of tools as a preliminary step in the development of tools designed to meet those standards (de Primio, 1986; Fickas, 1987; Fu, 1987; Keen, 1984), three performed comparative evaluations of different tools in the context of a particular domain problem (Beach, 1986; Wall, 1985; Waterman, 1982), and the remaining two references set up general guidelines to be used in the evaluation of tools (Richer, 1986; Waterman, 1986c). The eight articles dealing specifically with shells were in agreement on most features of shells; they differed primarily with regard to questions of generality vs. specificity. Articles written from the tool users perspective placed a premium on how well the tool matched the parameters of their problem, while those written by tool developers tended to value generality.

Problem definition. The first problem that must be dealt with is the question of what constitutes a shell. Although expert systems have been implemented in many languages, not all high-level languages constitute expert system shells. A shell must have some built-in facilities for lifting the burden of programming from the shoulders of the system developer, allowing him to concentrate on the knowledge acquisition process (Keen, 1984). Richer (1986) states that

> A shell...means an expert system development tool that provides standard ways for representing knowledge about a similar problem domain and task, and a standard inference procedure... All tools provide both a high

level language and an environment for creating, modifying, and testing systems (p. 167).

Software evaluation may be defined from the software scientist's perspective as "determining and evaluating in context the completion time and resource consumption of processes executing specified tasks on abstract machines" (Lynch, 1981, p. 172). To the user, however, the term is more broadly defined; performance is not only how much and how fast, but also how well the software solves the problems presented to it (Francis, 1981). Evaluation metrics based on these considerations are directed primarily at the end-user of the product and, although they deserve consideration, they are not sufficient for the evaluation of expert system shells. Here the main consideration is quite often ease of development rather than end-user performance; it often makes sense to use a shell for development, testing, and tuning of the expert system and then re-code the developed system in a more efficient language for delivery (Waterman, 1986c). Of course, given two shells with equal development efficiency, the tool which provides the more efficient runtime environment will win out.

The research confirmed some general propositions articulated in the planning of this project (Baker, 1987), namely that the evaluation of expert system shells is problematic in several respects. First, it is very difficult to make an evaluation of a shell independently of the context in which it will be used. While it is theoretically possible to solve any given AI problem using any of the standard representation schema and inference mechanisms, specific constructs have been developed for specific problems. These constructs are advantageous in that they provide the most natural approach to the construction and manipulation of domain knowledge (de Primio, 1986). The presence or absence of a particular feature may have a significant impact on the applicability of a given shell towards a given problem. Given a task, it is true that there may be a number of tools equally well suited to the task, but it is also true that none of the tools will be perfect for the task (Waterman, 1986c).

Secondly, the expert system shell is a meta-system, and as such must serve many masters. It must have the representational power to comfortably capture domain knowledge and the logical power to use that knowledge. A good shell system must be "builder friendly," both to the trained knowledge engineer and to the possibly naive domain expert, and have the necessary tools to construct a natural interface for the end-user of the

product (Keen, 1984). Finally, more standard software evaluation issues such as cost and time efficiency, hardware requirements, and availability of support must be dealt with.

Categories of criteria for evaluation of shells. The criteria for evaluation of expert system shells may be broken down into five broad categories.

- Basic features: structures for encoding knowledge, control and inference mechanisms, extensibility of built-in features, generality, programmability, ability to link to external hardware and software, explanation and hypothetical reasoning capabilities.

- Development environment: support for programmers or knowledge engineers during development, testing, and tuning of the knowledge base, appropriate interface for both the knowledge engineer and domain expert.

- Support: availability of training, quality of documentation, phone support for answering technical questions.

- Cost factors: cost of software and required hardware, overhead of developing trained knowledge engineers.

- Functionality: ease of learning, ease of use, efficiency of development and run-time environments, range of applications, and performance characteristics of the tool.

Of these categories the first four may largely be measured by a description of the features and the price of the tool, although the cost dimension includes facets not evident until the tool is applied to the target problem, mainly related to the time-on-task; the last category, functionality, is where an evaluation of the performance of the tool needs to be taken into consideration.

Basic features.

This area encompasses issues of features of the shell itself that are available to the knowledge engineer. The presence or absence of a given feature may have a significant impact on the utility of a shell for attacking a specific task. However, the inclusion of

numerous features may be a double-edged sword in that the learning curve for learning to use the tool adequately may become quite steep.

Specialized vs hybrid systems. All sources are in agreement that at the very minimum, an expert system shell must include mechanisms both for structuring knowledge and for inference based on that knowledge. The earliest expert system shells (i.e. EMYCIN and KAS) were developed from specific expert systems by emptying domain dependent knowledge bases, leaving the structures and inference mechanisms developed to solve the original target problem (Waterman, 1986c). Such shells necessarily lack the flexibility of newer hybrid systems which provide a variety of structures for representing knowledge, a choice of inference mechanisms, and often allow the knowledge engineer to define new representation and control mechanisms (Richer, 1986). It is an open question as to whether these hybrid features are desirable or not.

Developers of expert system tools rate generality very high on their list of criteria for a good shell. In a 1986 article on BABYLON, a hybrid shell under development at that time, the author states that although hybrid systems do not inherently have greater representational power than pure systems, their chief advantage may lie in a capacity to represent domain knowledge in a more "natural" fashion (de Primio, 1986). The same study goes on to suggest that many problems could prove amenable to a distributed problem solving approach, using different formalisms to attack segments of the same problem. In this type of context a hybrid system with a variety of knowledge schemata and inference mechanisms could greatly enhance the representational power of the shell.

The increase in flexibility offered by hybrid systems does not come without cost attached. Donald Waterman (1982), in a pioneer study done by the Rand corporation, cautions

> The more general the control and representation, the more time-consuming and difficult is the representation of any particular knowledge because of the excessive degrees of freedom... By contrast, the more rigid and constrained the control and representation, the easier it is to represent knowledge--if that knowledge can be represented at all within the limited representation paradigm (p. 45).

Hybrid systems with their great flexibility may be much more difficult to learn and use, and in some applications a simpler tool with more structure and less options may prove to be more cost effective (Richer, 1986). Not all users of expert system tools have

the same needs. For the professional knowledge engineer already trained in AI methodologies and involved in the construction of expert systems across a broad spectrum of contexts, the time spent learning to effectively use a powerful, general purpose tool with a wide range of application would be time well spent. This situation is in sharp contrast to that of a small firm interested in developing a particular application in-house using its own personnel; here, the time and cost effective solution might be to use a tool more specifically designed for the task at hand.

With regard to the evaluation of the basic features of the language it will suffice to describe the features present in a given tool. The question of which features are included will have a profound effect on the functionality and range of application of the tool.

Language features. An important dimension of the evaluation of expert system tools involves the characterization of tools based on the presence or absence of standard schemata for representing knowledge and making inferences. The possible parameters of knowledge representation include:

- Support for representing basic components of symbolic knowledge such as facts, definitions, heuristics, and procedures for performing a task or reaching a goal (Richer, 1986).

- The types of formalisms available for representing that knowledge, such as predicate calculus, logic programming, rules, semantic networks, and frame-based structures with slots and values (Richer, 1986; Wall, 1985).

- Inference and control systems providing support for pure deductive inference, forward- and backward-chaining inference engines, and class inheritance (Richer, 1986).

- More sophisticated inheritance structures providing both system- and user-defined inheritance mechanisms, hierarchical structures, and availability of demons for access-oriented applications (Wall, 1985).

- Object-oriented programming and message passing systems (de Primio, 1986; Waterman, 1986b).

- Blackboard systems for more complicated applications involving cooperating knowledge sources (Waterman, 1986c).

- Extensibility based on programmability in a high level language and the ability to link to external software and hardware (Fu, 1987).

- The ability to provide hypothetical reasoning capabilities and explanation facilities at the end-user level (Fu, 1987; Wall, 1985).

- A "Very High Level Language" for encoding knowledge in a form which is more accessible to domain experts, thus allowing them to participate more fully in the knowledge acquisition process (Keen, 1984).

- Incremental compilation of the knowledge base and an interpretative mode to be used during the knowledge acquisition phase (Beach, 1986).

- Conflict resolution strategies for rule based applications (Beach, 1986).

Evaluation in this area would consist of a checklist noting which of and to what degree these features are built in to the given tool and which are readily available as user-defined extensions to the tool. The types of features provided by a shell package will significantly affect the performance and functionality aspects of evaluation.

Development environment. The development environment refers to the explicit features available to support the programmers or knowledge engineers during the development process. As expert system tools attain greater sophistication this dimension takes on greater importance. Waterman wrote in 1982, "Other very useful, although not indispensable, features to incorporate into tools are facilities for explanation, interaction, and local operating system accessibility" (p. 276). In 1986 the same author writes, "The more extensive the support facilities, the more attractive the tool becomes as a development vehicle" (Waterman, 1986c, p. 144). Given a tool with adequate representational ability to handle a given problem, "...then it is likely that a large amount of the time on a project will be spent building the knowledge base and refining it...explicit support for knowledge acquisition is critical" (Richer, 1986, p. 169). An extensive and yet easy to learn and use developers interface can significantly decrease the amount of time and effort spent in this critical phase of development. Desirable features of the development environment include:

- Knowledge base editors to assist in the knowledge acquisition process. Desirable features for inclusion into knowledge base editors include spelling checkers, syntax checkers, tools to check the knowledge base for consistency and redundancy (Richer, 1986), and knowledge extraction capabilities in which the editor assists the user in entering new knowledge into the data base (Waterman, 1986a).

- Graphical builder interfaces providing for the display of hierarchical relationships, support in editing the knowledge base, and aid in testing and debugging (Wall, 1985; Richer, 1986).

- Debugging and validation aids such as trace features, break packages, facilities for storing and batch processing a large set of test cases (Waterman, 1986a), displays of reasoning paths and order of rule firings (Wall, 1985), automatic bookkeeping indicating by who and when the knowledge base was updated, and explanation and hypothetical reasoning facilities at the knowledge acquisition level (Waterman, 1986a).

- Input/output facilities directed at the end-user of the system including procedures for menus, windows, and writing i/o routines, the ability to provide runtime acquisition of knowledge (Waterman, 1986a), and integration with existing systems such as video-discs, natural-language interfaces, sound-generation and voice recognition packages (Richer, 1986).

Evaluation in this area will tend to be subjective with regard to what features are included in the shell and how they are implemented, just as choice of a word processing system often boils down to what the personal preferences of the users are; however, the features provided by the development environment may have a measurable effect on the overall performance of the package.

Support. Support as used here refers to external support systems for the tool user rather than the support for developers provided by the features of the tool. Significant questions include:

- Maturity of the tool. Is the tool fully developed and debugged with a large user community and good reputation? Expert system tools may be classified as experimental, research, or commercial tools. Only the last classification should be highly recommended for the average tool user who is not interested in having to perform maintenance on the tool (Waterman, 1986b).

- Maintainabililty. Is the tool developer still interested in the maintenance of the product? Is assistance in maintaining the tool readily available? (Waterman, 1986c).

- What is the quality of the documentation? Is it complete and understandable? Is online documentation and updating available? Is source code available and what does it cost? (Richer, 1986).

- What kind of training is available? Will the vendor provide on-site training? Are training locations accessible? (Richer, 1986).

- what kind of services are available through the vendor? Will the vendor make custom modifications to the tool if required? Does the vendor offer knowledge engineering services? (Richer, 1986).

Cost. The easiest parameter to identify here is the cost of the software package itself. However, this basic cost is not the only, and perhaps not even the most significant, factor in an assessment of the cost of an expert system shell. Requirements for high-powered hardware setups, the cost of training for personnel involved, and hidden costs related to the ease of use of the shell (and it's impact on the time required for development) may add considerably to the expense of implementing a shell. Startup costs must be balanced against the mileage expected from the developed product; a tool with a high overhead margin may be justified only if the target application is to find wide distribution or extensive use (Richer, 1986). If the tool is being used to develop marketable applications, then the cost of licensing agreements for runtime applications must also be considered.

Functionality. Evaluation of the functionality of the expert system shell moves beyond a simple description of the basic features and development environment of the shell. The issue at hand is how well and how cost-effectively the software performs the

task which it was designed for; the performance characteristics of the shell are, of course, a function of the language features and development environment, as well as cost and support factors. Factors to consider in the assessment of functionality are:

- How difficult it is to learn and use the tool (Richer, 1986).
- Hardware requirements, both at the development and end-user levels (Richer, 1986).
- Robustness of the tool and developed applications, error-handling abilities at development and end-user levels, feasibility of running systems in buggy form (Richer, 1986; Fickas, 1987).
- Efficiency of development and runtime environments.
- Range of application of the tool (Richer, 1986).
- Performance characteristics of the shell.

Performance metrics for shells. The performance of expert system shells cannot be measured using only traditional software metrics such as dhrystones/sec or speed of compilation; much more significant is the amount of time involved in designing the structure of the knowledge base and in the process of knowledge acquisition. This time is a function of both the development environment and the language features available in the package. Any methodology for evaluating expert system shells must take into consideration how well the package fits the problem at hand. With regard to the evaluation of development tools"...a key component is a side-by-side comparison trying to solve the same problem with each tool" (Wall, 1985, p. 280). This was the approach taken in the reviewed studies which attempted to move beyond simple descriptions of the features of a tool and make an evaluation of the performance of the tool in a working situation (Beach, 1986; Wall, 1985; Waterman, 1982). A problem was selected and the shells under consideration were used in attempts to solve the problem. Although this approach may be sufficient for an assessment of which tool to use for a specific problem, it is certainly not sufficient for the evaluation of a tool which is represented as a general-purpose tool.

Researchers point out the current deficiencies in the realm of evaluation of general purpose tools. According to one researcher, "AI research has not yet progressed to the point where we can list in the abstract all possible...[tool] features ... and show how they map onto or suggest problem and application features" (Waterman, 1986c, p. 146). Richer (1986) writes

> continued research studies should help us classify problem types and problem-solving methods according to some dimensions that would indicate which representational frameworks are more appropriate for a given problem using a particular problem-solving method (p. 168).

These statements suggest the need for a set of standard problems and problem-solving methodologies to use in the evaluation of tools. Statistics could be compiled on the time and effort required to solve each of the standard suite of problems, as well as the investment of time required to learn to use the system. Implicit in this schema is a classification framework for shells based on their performance in each of the problem areas.

Conclusions. It appears that the task of evaluating expert system shells may be broken down into two main sub-tasks. One sub-task is primarily descriptive in nature, consisting of listing the language and user-interface features of the system, identifying the available vendor support, and focusing on the more obvious cost factors involved in the basic use of the system. Evaluation in this realm is quite mature and easily accessible, with numerous studies having been done covering a wide range of products. Gilmore, 1986; Harmon, 1985; Walker, 1986; and Waterman, 1986d are examples.

The second sub-task is much more subtle and much less well-developed and well-defined. That task is the evaluation of the functionality of the shells, and it is a difficult task in at least two respects. First, it includes an assessment of the effects of the language and development features on the performance of the shell, a dimension which is heavily dependent on the particular problem context chosen. Secondly, the evaluation of functionality from the tool users perspective must include an estimation of the hidden costs inherent in the system. This assessment may even be more difficult than the first since it includes not only domain dependent dimensions, but also considerations specific to the individual user, such as the availability of trained staff. It is in this area that continued research and a standardization of methods is advisable.

A Functional Approach to the Evaluation of Expert System Shells.

Rationale. As detailed above, the task of evaluating expert system shells is a problematic domain. Some of the approaches to this problem have included catalogs of shells and their features (i.e., Liebhaber, 1987) and direct comparisons of shells used to solve the same problem (Beach, 1987; Waterman & Hayes-Roth, 1982; Richer, 1986). The first approach is primarily descriptive and is unlikely to be of much utility for many potential tool users since in many cases the potential user is unaware of what features to look for in a tool or how those features will interact with the development environment at hand. The second approach, concentrating on the functionality of the tool, is less well-developed. One type of functional approach, the case study method, holds more promise in that if properly done it provides some insight into the performance of a tool in a working situation.

Perhaps the most common purpose of an evaluation of expert system shells is to assist in the selection of a shell or shells for a project, series of projects, or long-term use by an organizational unit. Here it is necessary to match the capabilities of the shell to the requirements of one or more problem settings, with their associated organizational and programmatic constraints. The nature of the problems and constraints will vary depending on organizational focus and resources. Expectations of shells to be used in research will likely differ from those of shells intended to facilitate product development. Evaluation of long-term needs may use different methods or obtain different conclusions than evaluations which consider only short-term needs. A library of cases could be searched by such problem setting characteristics to find similar situations and the shells used to tackle them, along with process and product data. In this manner, a shell well-matched to the problem characteristics can be selected.

In recent workshops for both tool developers and tool users at the Rand Corporation (Rothenberg, et al., 1987) the case study method was espoused by tool developers as an attractive and viable methodology for evaluation of shells; one of the results of the workshop was a recommendation that an agency be created to collect and catalog case studies on the use of expert system shells. The enthusiasm for case studies was not shared by the tool users, however, who tended to take a much more practical view. For the users, issues such as the cost involved in conducting case studies, difficulties in studying secret or proprietary projects, and the problems of obtaining and analyzing data

were of paramount importance. Their objections were not founded upon any inherent weaknesses in the case study method, but rather on their perceptions of the difficulty of conducting usable case studies in a relatively unobtrusive manner. It is the contention of this paper that despite these objections, the case study can be an extremely valuable resource in the evaluation and selection of expert system shells given the proper methodology.

In order for case study data to be usable, considerable refinement and systematization of the case study method is required. The methodology must be unobtrusive and capable of being administered internally for those situations in which the context is sensitive, while still being powerful enough to get at the important characteristics of the case without compromising security. The relevant aspects of the case study need to be clearly defined, captured, reported, and catalogued so that potential users can better match case studies to their particular requirements. This paper examines some of the issues involved in effectively using case studies in the evaluation of expert system shells. In this treatment we will critique the case study as an evaluation instrument, and work to provide a systematic framework on which to build future case studies.

Approach to the problem. The first task will be the identification of some of the variables which must be dealt with in a case study; these include considerations about the problem domain, the domain expert, the knowledge engineer(s), the observer, the data collection process, analysis of the data, and the reporting of results. The objective of this process will be to develop a systematic methodology for dealing with such variation between case studies in a packageable form which would yield replicable results. Discriminating methodological constraints are provided by the necessity of providing a package which will:

- not induce resistance to its' implementation by tool users.
- be capable of wide-scale application by tool users themselves with minimal intervention by outside personnel.
- provide data which captures the functionality of the tool without compromising classified or proprietary details about the context of the problem .

Where applicable, illustrations will be provided from two expert system development projects carried out by this project in an effort to gain some insight into the problem of evaluating shells. In both development efforts the task was the same, to develop a system, the CRT-Adviser, that would advise users on the appropriate indices and methods for assessing the reliability of criterion-referenced tests. The first development effort involved the elicitation of domain knowledge from an expert by a knowledge engineer, and the construction of an expert system using the shell M1™ on an IBM-XT™. In second effort, videotapes, field-notes, and the product of the first project were used to construct a parallel system using the shell NEXPERT™ in a Macintosh™ environment.

General considerations

Problem domain. Most existing case studies are in the form of comparisons of tools used to solve the same problem (Waterman & Hayes-Roth; 1982, Beach, 1986; Wall, 1985), but these types of studies are limited in that they are restricted to 'toy' domains which may have limited correspondence to real-world problems. A more effective approach would be to study actual problems, but this approach imposes difficulties in comparing performance of shells across different problems. Research is currently being conducted on the classification of problems and the development of a taxonomy of problem types (Chandresekaran, 1983, 1986), but at present there exists no really workable schema for accomplishing this end.

One necessary component of a case study, therefore, is a thorough description of relevant aspects of the problem domain. This information could be gathered in the form of a questionnaire; some of the relevant dimensions include the type of problem (i.e., diagnosis, consultation, etc.), the kinds of knowledge structures embedded in the problem, types of inference and control mechanisms indicated by the problem, and the user interface requirements. It might be interesting to elicit this information both at the pre and post production phases; differences between these measures might provide some metric for how much the development process is shaped by the choice of a particular tool. In fact, a structured instrument for the acquisition of this type of information is likely to be an asset in the knowledge elicitation process since the solution of the problem requires that characteristics of the task space be made explicit.

Olsson and Rueter (1987) describe an assortment of indirect methods, such as multi-dimensional scaling and Johnson hierarchical clustering, which could potentially be

used by the knowledge engineer during the knowledge acquisition process. While these indirect techniques are somewhat limited in their ability to describe the objects in the domain, they are advantageous in that they highlight the relationships among those objects. The inclusion of a standard set of indirect methods in a case study package could then be beneficial in two respects: first, indirect techniques would be valuable in instances where the context is sensitive by providing a picture of the relationships between domain objects without compromising security, and secondly, the output from these methods could be of assistance to the development team and could potentially provide some insights into the structure of the problem that may not have been easily accessible through the use of more direct methods. The additional effort required could be offset by possible gains in efficiency, thus tending to reduce resistance by the development team to the implementation of this methodology.

Domain expert. Another significant source of variation in the case study method is the amount of expertise regarding expert systems and AI paradigms possessed by the domain expert. The knowledge acquisition process is a very demanding phase of the development process with respect to resources, and it is likely that a domain expert already familiar with knowledge engineering techniques in general, or a tool in particular, could help to speed this process. Although the primary direction of transfer of information in the knowledge engineering process is from the domain expert to the knowledge engineer/system, the format of the questions asked by the engineer may be sufficient to cause some reverse transfer of knowledge engineering expertise. Another possibility is that the expert may become more proficient in the spontaneous decompilation of domain knowledge through his participation in the knowledge extraction process.

These processes are illustrated in the knowledge elicitation process for the CRT-Advisor. The domain expert used in this project was initially quite naive with respect to knowledge engineering techniques. In addition, considerable prompting was required by the knowledge engineer in order to assist the expert in the "decompilation" of his domain knowledge into a form usable by the engineer. The shell used in this project was M1™ which is a rule-based, forward-chaining system, and the inference mechanisms available in M1™ proved to be relatively evident to the expert. Within the first day of the knowledge elicitation process, the expert had adapted to the environment to the point where he could suggest workable rules for the system.

One consequence of the above considerations is that the degree of knowledge engineering expertise possessed by the domain expert becomes a significant variable to consider in the matching of case study data to the situations of potential users of that data. A domain expert that has participated in several prior expert system development efforts may greatly improve the efficiency of the knowledge elicitation process, and so relevant aspects of this dimension need to be captured.

Evaluation in this area would have a two-fold purpose. A pre-production instrument to obtain a profile of the expertise of the domain expert with regard to knowledge engineering could be used to allow users of the case study to match the profiles of experts used in case studies to the profiles of their own domain experts. This information could be obtained through a questionnaire and through a carefully designed criterion-referenced test, perhaps administered adaptively by a computer. Further research could be done to identify the salient domain features, and a criterion-referenced instrument could then be developed to measure each objective; since the results of this instrument will be used only for diagnostic purposes it could be quite short and still provide adequate reliability (Hambleton, 1984). This type of instrument would be unobtrusive enough to satisfy the tool user, and yet would provide sufficient information to assist in the classification of the case study.

The second purpose would be to provide a direct metric for the evaluation of the tool. The two effects mentioned above, whereby 1) the domain expert tends to become more of a knowledge engineer and 2) he attains readier access to his own compiled expertise, could provide a measure of the ability of the tool to educate the domain expert. The first effect could be easily measured by comparing performance on pre- and post-production administrations of the criterion-referenced instrument described above. The second effect is perhaps more difficult to measure as it may be highly domain-specific; examination of protocols of knowledge engineering processes would provide insights into this effect.

Knowledge engineer. One of the major concerns for any potential user of a tool has to do with the qualifications of the personnel available. Knowledge engineers vary greatly in their familiarity with the domain at hand, their proficiency with general knowledge engineering techniques, and their proficiency with a given tool. A development effort which runs smoothly given a team already proficient in the use of a sophisticated tool might become cost and time prohibitive given a novice engineer. Users of data from case studies

must be able to match the available case studies to their own situations, and so profiles of the personnel involved are a must for a case study.

The same methodology used to measure the knowledge engineering expertise of the domain expert could be used to obtain a profile of the knowledge engineering expertise of the development team. An appropriate battery of instruments would be a criterion-referenced test measuring familiarity and proficiency with general knowledge engineering techniques, and a questionnaire which inquires about familiarity with the particular tool being used and prior expert system development efforts and the characteristics of those efforts. Again, differences between scores on pre- and post- production administrations of the test would provide a metric for the educational value of the tool.

A more difficult proposition is the task of capturing the degree of domain expertise of the knowledge engineering team; any such evaluation would have to be capable of spanning diverse domains, a consideration which rules out the use of criterion measures which by definition are domain-specific. A more appropriate instrument would be a general questionnaire. In situations where the development team consists of more than one individual the level of expertise of the team would most appropriately be taken to be the level of the most proficient member of the team. In the development of the CRT-Advisor, both the knowledge engineer and the observer (as it turned out, a significant contributor to the project), were quite familiar with the domain, and this familiarity tended to speed the process with respect to the elicitation of domain expertise. This dimension probably does not interact directly with the characteristics of the shell, but would tend to bias metrics measuring the amount of effort spent in the knowledge elicitation phase.

Observer. Case studies may take two forms, either internal or external. External reviews have an advantage in that they may be done by trained observers versed in the collection of qualitative and observational data. Rothenberg, et al. (1987) found that tool users preferred the concept of external studies, but at the same time foresaw difficulties in situations where the context was sensitive enough to preclude the possibility of external review. It is possible that the prospect of external review may be attractive to the tool user at a surface level, but when applied to real projects may prove to be repellent for at least two reasons. The first is related to security. It is likely that many users in the private sector would be unwilling to submit their proprietary projects to review and publication. And second, the presence of an external reviewer may be perceived as an unwelcome intrusion by the personnel actually involved in the development process. Another problem

with external reviews is connected with the sheer magnitude of the task of collecting enough case studies to create a usable library of such studies. The cost factors involved in the collection of external reviews might prove to be prohibitive.

One possible way around this dilemma is to provide sufficient guidelines for the role of the observer so that this role becomes a skill accessible to in-house personnel. Some of the duties of the observer would be to collect and review transcripts of knowledge elicitation sessions, conduct pre-, mid-, and post-elicitation interviews of participants, and apply coding schemes to data collected in the course of the study. These considerations are covered at length later in this paper. Once appropriate methodology is identified for the collection and analysis of data there arises the possibility of providing appropriate training for in-house personnel in the application of that methodology.

With respect to the use of the case study as an experimental methodology, the experimenter must consider and control for the confounding effects of the presence of the observer on the knowledge engineering process. With respect to the use of the case study as a routine method of collecting information about the use of a tool to solve a real-world problem, however, those same influences could have a beneficial effect on the course of the project. It is conceivable, for example, that the knowledge engineering team could receive valuable insights into the progress of their project in the course of a mid-production interview by a sophisticated observer, or that analysis of protocols obtained during the knowledge acquisition phase would have positive effects on the future course of the project. A possibility which might tend to make the case study method more attractive to tool users then would be to not try to eliminate the effect of the observer on the process, but rather to ensure that the observer has a positive effect on the process; this type of effect could go a long way toward reducing the resistance of tool users to the case study method. The observer in this paradigm performs an additional function, that of facilitation of the knowledge acquisition process. This indeed was the role that the observer played in the development of the CRT-Advisor.

This approach adds another dimension to the case study methodology, namely that of obtaining a profile of the observer. The observer then remains one of the variables in the method, but one that is sampled rather than controlled. The same general framework which is used to obtain profiles of the domain expert and knowledge engineering team would be utilized to profile the capabilities of the observer, with appropriate modifications of the

domain objectives in light of the possibility that those characteristics which make one an effective facilitative observer belong to a unique domain.

Next arises the question of who is the best person to fill the role of observer. It seems quite likely that most development efforts already include a member whose duties would require little extension to take on the added role of observer, and in fact a person working in a supervisory capacity might find that the structure imposed by the observational process serves to support them in their supervisory capacity. Any development effort could benefit from the presence of an observer that provides constructive feedback during the course of the project, and so it would be worthwhile to train such and individual. This type of effect could again work to decrease resistance to the case study effort by the development team.

Data collection. Some of the types of data to be collected are enumerated in the sections above. They are:

- A thorough classificatory description of the problem domain.
- Pre- and post-elicitation profiles of the domain expert regarding the expert's level of sophistication in knowledge engineering techniques.
- Pre- and post-elicitation profiles of the knowledge engineer focusing on prior domain expertise, familiarity with general knowledge engineering techniques, and familiarity with the tool used for the project.
- A profile of the characteristics of the observer.

The data listed above are collected outside the framework of the actual knowledge elicitation process. During the elicitation process the primary sources of data will come from transcripts of the knowledge acquisition sessions, journals kept by participants in the process, interviews of participants by the designated observer, and logs of time spent by the participants (Slawson, 1987, 1988). The collection of these types of data will be primarily the responsibility of the observer.

Transcripts of the process could be obtained by either video or audio taping the knowledge elicitation sessions. While the use of video tapes might be too intrusive to be

acceptable to tool users in general, it is a well-established practice for knowledge engineers to audio tape their interactions with the domain expert for their own future reference, and so installing this practice as a standard part of the knowledge elicitation process should not meet with undue resistance. Verbatim transcriptions of the tapes would probably provide voluminous quantities of virtually unusable data, and so it would be advantageous to develop procedures whereby the observer could distill usable information from this data source.

Collection of the other types of data would be facilitated by the establishment of standardized forms and techniques. The proper construction of such instruments would involve the minimization of inconvenience to the development team and the maximization of the likelihood of side effects beneficial to the development process. The data collection process could then serve the ancillary purpose of providing ongoing monitoring of the development process.

Data analysis. The major analytical task internal to the case study method has to do with the analysis of process data. Transcripts of knowledge elicitation sessions can be analyzed on several levels and coded accordingly. Analysis can focus on problems encountered in the process, departures from an idealized model of the knowledge engineering process (Slawson, 1987), the types of interactions between the knowledge engineer and the expert (both quantitative and qualitative), and a task analysis of the knowledge acquisition process. Consistent coding schemes and guidelines for their application are necessary in order to make comparisons across case studies, as well as to assist personnel involved in the internal collection and analysis of data in those situations where external review is not a possibility.

The other major analytical task is external to the case study and involves the cataloging of a large body of case study information, as well as the development of facilities and heuristics for matching case study data to particular problems and development environments. Considerations with respect to this dimension are considered in the section on use of the case study data.

Use of the case study data. The purpose underlying the evaluation of expert system shells is that of providing information which can be used by the potential tool user in the selection of an appropriate instrument to solve a given problem in a given development

environment. Rothenberg, et al. (1987), developed a process to use in the selection of tools. The steps in the process are:

1. Determine application characteristics.
2. Identify relevant contexts.
3. Derive relevant tool capabilities.
4. Identify discriminating metrics and assessment techniques.
5. Identify available tools.
6. Filter available tools to identify candidate tools.
7. Prune and prioritize each framework dimension.
8. Apply the framework schema to evaluate and select tools.

The framework outlined in this paper overlays the first six steps in this process within the contexts of prototyping and development of an application.

The first step involves the determination of application characteristics; these include the requirements and characteristics of the particular domain as well as those of the development environment. This task would be accomplished through the use of the same instruments used to obtain profiles of the domain, domain expert, and the knowledge engineering team in the case study methodology. The resulting profiles would then be comparable to the profiles cataloged along with the case studies, as well as a source of valuable information to the tool user.

Once a profile of the application characteristics of the potential user is obtained, appropriate heuristics would be used to match this profile with those of case studies on file. The solution space could be pruned at this stage through the application of discriminating metrics, defined by Rothenberg, et al. (1987) to be constraints, such as budgetary or hardware restriction, which can be used to discriminate between tools at an early stage in the selection process. It is unlikely that exact matches will occur, and it is possible that the heuristics applied to make selections will depend to some degree on the application characteristics of the potential user. For example, if a potential user has a knowledgeable development team, then the weight assigned to the dimension of development characteristics would be less than that assigned in the instance of an effort involving a novice team. In general, the highest weight would go to the problem characteristics,

followed by the characteristics of the development team, domain expert, and observer in that order.

The output of the process described above would be the set of candidate tools referred to in step 6 of the RAND study (Rothenberg, et al., 1987). The objective of the case study framework is not to select the tool for the user; that decision is best left to the parties that must eventually pay the freight. The objective instead is to provide provide users with a manageable subset of the available tools from which to make their decisions based on their review of the data from the case studies.

A Sample Case Study Evaluation

This section describes the methodology and some highlights of the results from an evaluation of the prototype expert system. The expert system was the CRT-Advisor referenced earlier, and the shell used was the shell M1™ implemented on an IBM computer. Data came from observations, interviews, self-report, and other sources. Particular attention was paid to the ways in which qualities of the expert system shell determined the process and outcome of development. Concluding this section is a discussion of some benefits and pitfalls of the case study as a method of evaluating shells.

Participants and task characteristics. The participants included a domain expert, a knowledge engineer-programmer, and an observer-evaluator. The software being developed was a prototype expert system to assist in the process of assessing the reliability of criterion-referenced tests. The observer collected data for the evaluation before, during, and after the development effort.

The domain expert was a nationally recognized researcher and consultant in the area of educational measurement. He was also an experienced teacher, serving as a professor in a school of education. The development and workings of expert systems were new to him. The knowledge engineer-programmer was a graduate student in education who had taken only one course in measurement and evaluation; thus his knowledge of the domain was limited. He had some previous exposure to the shell used in the study, extensive experience with a related shell, and had developed several other expert systems in the field of education. The observer was very familiar with the domain of educational measurement,

but not highly experienced in the specific topic chosen for this project. He was also experienced in the knowledge engineering process.

Expertise in the selection of methods to determine the reliability of criterion-referenced tests is well-systematized and valuable, but not readily available. This task was selected for the test case for several reasons. For one, the prototype could be finished within a reasonable amount of time. Whether or not the actual task selected would be of use in other case studies, it is likely that case studies in other settings will also be limited in size. Another consideration in selecting the test case is that it was a consultation system and consultation is still the dominant style of interaction used in expert systems today. Since this type of problem is typical it should be easier to generalize what is leaned about the methodology to a variety of settings. The moderately powerful, PC-based shell and the specific task were selected because they met objectives of a real development project accessible to the evaluators. Although evaluation requirements of other hardware and software environments may vary considerably from those considered in this case, small systems are very common and are likely candidates for an evaluation.

[Insert Table 1]

Data collection process and materials. Data were collected on all phases of the development process, in the form of interview notes, written observation record, videotapes, and participant journals. The sources of data are described below:

1. A pre-elicitation interview between the knowledge engineer, the observer, and the project director was conducted. This interview served as an occasion to collect background information as well as a time to generate expectations for the project. The background data included an inventory of the knowledge engineer's skills and experience with the domain, the shell, knowledge programming, and related domains. Expectations for the shell and the knowledge engineering process were used by the observer as a point of departure during later observations and interviews. These expectations are summarized in Tables 1 and 2. The expectations served as a baseline to assist in noting deviations from expected progress, thus calling attention to possible interactions of shell capabilities and characteristics of the problem setting.

[Insert Table 2]

2. During the initial phases of knowledge engineering, the domain expert and knowledge engineer were observed as they interacted to do preliminary analysis and design for the test case. Later, the knowledge engineer was observed working with the shell and with the expert to build, test, and refine the system. The sessions were videotaped, and notes were taken to record observations and provide and index for the tapes.

3. Other data sources included written reports by the participants, the software itself, and post-interviews of participants. The knowledge engineer kept a dated journal in which he recorded his impressions of the process of using the shell to build the system. All domain expert and knowledge engineer notes, drawings, or other paperwork were labeled, dated and retained. The domain expert kept notes during the interviews on what he learned or thing he wanted to consider further. Any work files or significant intermediate versions of the software were also kept.

4. Between major steps of the software development process, and at the end of knowledge engineering sessions, interviews were conducted to correlate the various observations, ask probing questions, and record experiences of the participants. The interview was also an occasion to suggest and discuss prototypical metrics and standards for the variables identified as being significant for the test case. The data collection followed a two-step observation-interview protocol called *stimulated recall*, described below.

5. Following each step of development and each elicitation visit, interviews were conducted to help formulate new questions, evaluate hypotheses, and plan subsequent data collection. The overall process is summarized in Table 3.

[Insert Table 3]

Simulated Recall. To do a stimulated recall, the observer made notes during the videotaped sessions. The notes were intended to record comments, questions, and observations, as well as to serve as and index for the videotape of the process. The videotape included a continuous date and time stamp and each observation in the notebook was also accompanied by a note of when the event occurred. Videotapes and notes were then used at the end of each day to guide stimulated recall and analysis of the observed events. After allowing the knowledge engineer or domain expert to discuss the project in an open-ended interview format, the observer would replay selected portions of the tape.

The observer then asked question about what was being observed on the tape. Thus the observer had a chance to verify and clarify his understanding of what was observed earlier in the day.

During observations the observer was specifically looking for evidence of: 1) errors, difficulties, delays, problems, etc., which might be caused by weaknesses of the shell or by a mismatch of shell capabilities to the problem setting, and 2) knowledge engineering productivity boosts, positive participant attitudes, effective shortcuts, or improvements in quality which might be attributable to shell capabilities or features.

Approach to data interpretation. The notes and tapes were reviewed and observations cataloged. Categories for observations were developed and then were sorted so that relevant observations for specific questions were grouped together. The knowledge engineer's journal and the other reports were grouped together. The knowledge engineer's journal and the other reports were also carefully studied in light of the objectives of the study. The coding of observations in the original notes was reexamined after reading all the notes and forming new interpretations. Data were searched for both confirmatory and disconfirmatory evidence of intermediate interpretations.

[Insert Table 4]

Discussion. A description of the application characteristics is presented in Table 4. This is typical of the kind of information by which a user of a case study library might search for relevant cases. Examination of what helped and hindered the project, as well as interviews with participants resulted in identification of several shell capabilities that might have facilitated the process of system development in this case. These are features not present in M1™ that would be desirable for a similar project. Some highlights are presented in Table 5. Following this, some benefits and pitfall of using the case study method to evaluate an expert system shell are discussed.

[Insert Table 5]

Benefits and pitfalls of the case study methods. A number of benefits of doing the case study became apparent during the process. Perhaps the most valuable was the additional evaluation and learning that took place with the presence of an outside observer providing feedback and asking questions. The knowledge engineer reported that he learned

of new questions to consider when selecting a shell in the course of the study. He also reported that some of the observer's comments between knowledge engineering sessions brought conceptual issues to his attention sooner than they might otherwise have been, thus facilitating the process. Insight into the knowledge engineering process was gained. And ideas for improved shell design were generated. Rather than proving to be intrusive, the presence of an observer seemed to be positively motivating to the other participants. It should be noted, however, that the observer was already well acquainted with the other participants and with their abilities and responsibilities pertaining to the project. Under less ideal conditions, an outsider might have been more of a hindrance.

Another positive outcome of the study was the detailed description of the problem setting, the shell, and the process of using it on a real expert system development problem. Only a portion of the data is represented here, however, and the selections may not anticipate the questions of future evaluators. It may be that secondary users of a case study will need direct access to the source data in order to find the information that is of most interest to them. In this case the study was limited to the initial prototyping stages of development, and in other cases it is also likely that evaluators will want to focus on some stage of the development cycle that is of particular interest to them. The intended uses of a shell will dictate which phases of the development cycle give rise to the most uncertainty and hence the greatest need for detailed evaluation. In a system that will require long term maintenance and updating, such factors as truth maintenance and extensibility become the foci of attention, hence data collection could emphasize later phases of the development effort.

One of the disappointments of the study was the failure to generate useful quantitative metrics. Although the focus was intended to be on qualitative data, it was hoped that such indicators as 'rate of progress' (in number of rules, person-hours, or stages of development) would provide meaningful data on shell performance. Unfortunately, the rate of progress in this case was so much determined by schedules and convenience that it was impossible to make any conclusions about the progress or ease of shell used in terms of the overall effort or time expended. Even if the time and effort could be controlled, it is apparent that the number of rules required for a given level of system performance will vary depending on such factors as programmer skill, as suggested earlier. Of course, there is the possibility that a more ethnographic treatment (as in this presentation) might be of more value to potential users of case study data than a collection of descriptive metrics of dubious reliability and generalizability.

Further Research

The bulk of the work required to implement this model as a workable methodology will likely revolve around the dimensions of the characterization of the problem domain and the analysis and reporting of data. In a relative sense, the tasks of obtaining profiles of the development team members is small and well-defined, and can probably be entrusted to a small expert system. Larger policy issues revolve around the questions of what agency would be responsible for the administration of such a program.

Table 1

Expectations for the shell M1™

Some expected strengths	Some expected limitations
Pattern matching capability	Restricted to monotonic reasoning
Structured selection easy to implement	Lack of a procedural language
Easy enumeration of legal values in menus	Certainty factors increase the complexity of the program
Presumption function (meta-rules)	Difficulty in dealing with data intensive tasks

Table 2

Expected steps of expert system prototype development

Step 1:	Knowledge engineer reviews the background text and cases for understanding.
Step 2:	Overview. Get expert's terminology for any categories or taxonomies he uses, generally identify the kinds of input and output for the consultation process.
Step 3:	Have expert explain the cases so knowledge engineer can understand the inference and consulting process that goes on during a typical consultation. This is at the level of identifying possible stages in consultation, overall sequence of question types, etc.
Step 4:	Continue to develop ' strawman design'. Determine basic functions for the application, knowledge, and data sources. Create a block diagram of consultation process.
Step 5:	Map cases onto block diagram. Do as many as time permits. Begin to elaborate and refine the rough model to deal with specific, sometimes special, cases.
Step 6:	Revise block design based on Step 5. Collect control rules. Finalize scope of task.
Step 7:	Domain knowledge acquisition. Collect decision rules for knowledge base.
Step 8:	Begin initial coding of prototype.
Step 9:	Continue knowledge acquisition, coding, testing, until prototype is ready for further evaluation.

Table 3

Data collection process

Before knowledge engineering begins:

* Pre-interview to develop expectation s for development process.

For each day of knowledge engineering:

* Observations and video-tapes

* Notes and questions on differences between observed and expected behaviors

* Stimulated recall

Between each development step:

* Post-interview / analysis

* Formulate new expectations, hypotheses, and questions

Table 4

Application characteristics

Task or problem type:	Selection
Interaction / Interface style:	Dialog consultation via menu selection
Knowledge representation:	Rule-based with uncertainty
Size of problem:	Small
Inference and control:	Forward chaining, essentially monotonic reasoning

Table 5

Desirable shell capabilities not found in M1™

Editor features	Syntax directed editor Macro facility for variable names Run-time text modification to allow expert to modify text only
Knowledge-base interface	Multiple views of knowledge base Selectable kinds of debug / trace outputs Selectable user dialog options
Knowledge processing	Procedural language Consistency checking

References

Baker, Eva L. (1987). Artificial Intelligence Measurement System, Briefing Charts, ONR Contractors' Meeting, Yale University.

Beach, S. (1986). A comparison of large expert system building tools. The Spang-Robinson Report, 2(10), 1-8.

Chandrasekaran, B. (1983). Towards a taxonomy of problem solving types. The AI Magazine, 4(1), 9-17.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. IEEE Expert, Fall, 1986, 22-30.

de Primio, F. (1986). An integrated tool for building expert system - state and future requirements. In T. Bernold (Ed.), Expert systems and knowledge engineering: essential elements of advanced information technology (pp. 187-191). Amsterdam: Elsevier Science Publishing Company.

Fickas, S. (1987). Supporting the programmer of a rule based language. Expert Systems, 4(2), 74-87.

Francis, I. (1981). A scientific approach to statistical software. Chapter 9 in A. Perlis, F. Sayward, and M. Shaw (Eds.), Software metrics: an analysis and evaluation (pp. 162-181). Cambridge, MA: MIT Press.

Fu, L. (1987). Ejaundice: a general purpose expert system building tool. In J. Gilmore (Ed.), Proceedings of SPIE - The international society for optical engineering, Volume 786: Applications of artificial intelligence (pp. 274-281).

Gilmore, J. F. & Howard, C. (1986, November). Expert system tools for practitioners. Paper presented at the First Australian Artificial Intelligence Conference, Melbourne, Australia.

Hambleton, R. (1984). Determining test length. Chapter 6 in R. Berk (Ed.), A guide to criterion-referenced test construction (pp.144-168). MD: Johns Hopkins Press.

Harmon, P., & King, D. (1985). Commercial tools. Chapter 8 in P. Harmon, & D. King (Eds.), Expert system: Artificial intelligence in business (pp.92-133). New York: John Wiley and Sons.

Keen, M. & Williams, G. (1984). Expert system shells come of age. In M. A. Bramer (Ed.), Research and development in expert systems (pp.13-21). Cambridge, England: Cambridge University Press.

Korf, Richard, Personal Communication, 1986.

Liebhaber, M. (1987). A survey of expert system development tools. Contract #MDA903-86-C-0210 for US Army Research Institute for the Behavioral Sciences. Fort Leavenworth, KS.

Lynch, W.C., & Browne, J.C. (1981). Performance evaluation: A software metrics success story. Chapter 10 in A. Perlis, F. Sayward, and M. Shaw (Eds.), Software metrics: an analysis and evaluation (pp. 182-197). Cambridge, MA: MIT Press.

Olsson, J., & Rueter, H. (1987). Extracting expertise from experts: Methods for knowledge acquisition. Expert Systems, 4(3), 152-169.

Richer, Mark. (1986). An evaluation of expert system development tools. Expert Systems, 3(3), 166-181.

Rothenberg, M., Paul, J., Kameny, I., Kipps, J., & Swenson, M. (1987a). Evaluating expert system tools: A framework and methodology. Contract #MDA903-85-C-0030, R-3542-DARPA. Santa Monica, CA.

Rothenberg, M., Paul, J., Kameny, I., Kipps, J., & Swenson, M. (1987b). Evaluating expert system tools: A framework and methodology - workshops. Contract #MDA903-85-C-0030, RAND Note N-2603-DARPA. Santa Monica, CA.

Slawson, D. (1987). Methods and metrics for the evaluation of expert system shells. Unpublished manuscript. Los Angeles: UCLA Center for Technology Assessment.

Wall, R. (1985). An evaluation of commercial expert system building tools. Data and Knowledge Engineering, 1(4), 279-303.

Walker, T. C., & Miller, R. K. (1986). Tools for building expert systems. Chapter 3 in Expert Systems. Madison, GA: SEAI Technical Publications.

Waterman, D. A., & Hayes-Roth, F. (1982) An investigation of tools for building expert systems. RAND Report R-2828-NSF. Santa Monica, CA.

Waterman, D. A. (1986a). The nature of expert system building tools. Chapter 8 in D. Waterman, A guide to expert systems (80-94). Reading, MA: Addison-Wesley Publishing Company.

Waterman, D. A. (1986b). Stages in the development of expert system tools. Chapter 9 in D. Waterman, A guide to expert systems (95-111). Reading, MA: Addison-Wesley Publishing Company.

Waterman, D. A. (1986c). Choosing a tool for building expert systems. Chapter 13 in D. Waterman, A guide to expert systems (142-151). Reading, MA: Addison-Wesley Publishing Company.

Waterman, D. A. (1986d). Catalog of expert system tools. Chapter 28 in D. Waterman, A guide to expert systems (339-365). Reading, MA: Addison-Wesley Publishing Company.